SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

WEIGHTED AND PRIORITIZED TASK SCHEDULER

Background of Invention

[0001] Digital signal processors (DSP) are widely employed to process data streams (frames) and/or tasks. A DSP in a voice gateway often processes data streams with various frame rates (e.g. five milliseconds (ms) to thirty milliseconds) and with a wide range of processing requirements (e.g. different echo canceller tail lengths, different codecs, etc.).

[0002] Figure 1 illustrates the general approach to processor task scheduling. All the tasks are placed in an execution queue 102 as they are generated. Each processing unit 104 (e.g. a processor), when available to process, checks the execution queue 102 to retrieve any available task and then executes the task. A task may be represented as an identifier corresponding to a data frame(s), data block, or other unit of information to be processed by the processing unit 104.

[0003] With this approach, there is no concept of processing priority between queued tasks. Tasks are merely processed in the order in which they are received by the execution queue 102. This type of task execution scheme may cause data flow bottlenecks under certain conditions.

[0004] For example, Figure 2 illustrates a shared execution queue 202 receiving two concurrent data streams 204 and 206. A first data stream of thirty-millisecond data frames (thirty milliseconds representing the time in between data frames in the stream) (frame stream A 204) and a second data stream of five-millisecond data frames (frame stream B 206) are queued in the shared execution queue 202, as each data frame arrives, by a frame processing scheduler 208. If the thirty-millisecond data frame A (first to arrive) is processed before the five-millisecond data frame B (second

to arrive), this could lead to a bottleneck for the data flow of the five-millisecond frame stream B. For instance, if the thirty-millisecond data frame A takes five milliseconds or more to be processed, then data frame B would not be processed until after data frame C (the next five-millisecond data frame in frame stream B) arrives.

[0005] Figure 2 illustrates a configuration of a shared execution queue 202 that may be accessed by multiple processing units 212 and 214 to process queued tasks.

[0006] As more data streams are processed through a single shared execution queue, the likelihood of data flow bottlenecks or congestion increases.

Brief Description of Drawings

[0007] Figure 1 is a block diagram illustrating one embodiment of a conventional shared execution queue processing tasks.

[0008] Figure 2 is a block diagram illustrating possible data flow bottlenecks for one embodiment of a shared execution queue servicing data streams of different frame rates.

[0009] Figure 3 is a block diagram illustrating a scheduling system with multiple priority queues and a shared execution queue according to one embodiment of the invention.

[0010] Figure 4 is a block diagram of various priority queues, a switch, and a shared execution queue illustrating fair queuing of new tasks according to one embodiment of the invention.

[0011] Figure 5 is a block diagram of various priority queues, a switch, and a shared execution queue illustrating fair and weighted apportionment/queuing of new tasks according to one embodiment of the invention.

[0012] Figure 6 illustrates various other ways in which new tasks may be queued in a shared execution queue according to a fair and weighted scheduling scheme.

[0013] Figures 7 and 8 are block diagrams illustrating a couple of ways in which tasks from one priority queue may be used to fill-in for a shortage of tasks from another priority queue in a fair and weighted scheduling scheme.

[0014] Figure 9 is a block diagram of illustrating how a priority queue may be skipped in a fair and weighted scheduling scheme when the priority queue does not have any more tasks to be processed.

Detailed Description

[0015] In the following detailed description of various embodiments of the invention, numerous specific details are set forth in order to provide a thorough understanding of various aspects of one or more embodiments of the invention. However, one or more embodiments of the invention may be practiced without these specific details. In other instances, well known methods, procedures, and/or components have not been described in detail so as not to unnecessarily obscure aspects of embodiments of the invention.

[0016] In the following description, certain terminology is used to describe certain features of one or more embodiments of the invention. For instance, "frame" includes any block or arrangement of data or information. The term "information"is defined as voice, data, address, and/or control. The term "task identifier" includes any representation corresponding to a unique data frame.

[0017] One aspect of an embodiment of the invention provides a method, system, and apparatus having multiple execution queues of different priorities coupled to a shared execution queue to provide the shared execution queue with available processing tasks. Another aspect of an embodiment of the invention provides fair and weighted execution of the prioritized processing tasks.

[0018] Figure 3 is a block diagram illustrating a scheduling system with a shared execution queue according to one embodiment of the invention. Note that the components of the system illustrated in Figure 3 may be implemented as hardware, software, and/or a combination thereof. As data frames are received, a frame processing scheduler 302 schedules each frame for processing.

[0019] In one implementation, each received data frame is assigned a task identifier, any representation corresponding to the unique data frame. Hereinafter, the term "task" is interchangeably employed to refer to a task identifier. The frame processing scheduler 302 may include a task scheduler which schedules new tasks, corresponding to

received data frames, for processing.

[0020] In one embodiment, a task priority scheduling scheme is employed to permit scheduling each new task according to the processing requirements for the corresponding data stream. One aspect of an embodiment of the invention classifies each task according to their processing priority.

[0021] In the implementation shown in Fig. 3, a demultiplexer (task router) 304 is employed to place each new task in one of various queues, each queue corresponding to a different processing priority. For instance, a High Priority Queue 306, a Medium Priority Queue 308, and a Low Priority Queue 310 are illustrated in the embodiment in Fig. 3.

[0022] While priority queues are illustrated as separate queues (e.g. 306, 308, and 310), these queues may also be implemented as a single queue segmented into multiple queues of various processing priority levels. In other implementations, a larger or smaller number of queues may be employed, the different queues to hold new tasks of different priority levels. In yet another implementation, each priority queue may hold new tasks in a range of priority levels. That is, a priority queue may hold new tasks of different degrees or levels of priority, but within a particular range of priority levels, with no overlap in priority levels between the different queues. In various implementations, the size of the priority queues (e.g. 306, 308, and 310) may be the same or vary according to the desired implementation. In one embodiment, the size of one or more of the priority queues may be dynamically changed according to the requirements of the implementation.

[0023] Task priorities may be assigned in a number of ways. In one embodiment, illustrated in Fig. 3, new tasks are assigned a priority level based on a lookup table 312. That is, each type of session and/or data stream is mapped to a priority queue. Thus, the lookup table 312 enables the demultiplexer 304 to place new tasks, belonging to a particular session or data stream, in the appropriate priority queue. In one implementation, the look-up table 312 is communicatively coupled to the frame scheduler 302 so that as new tasks are sent to the demultiplexer 304 the look-up table 312 provides the corresponding priority level/classification to the demultiplexer 304 to enable it to route the new task to the appropriate priority queue.

[0024] Priority classifications may be pre-assigned and/or dynamically assigned based on various factors including frame size, echo canceller tail length (in systems where an echo canceller is employed), codec type, frame processing requirements, etc. According to one implementation, the look-up table 312 is pre-configured with default priority assignments.

[0025] Where a dynamic priority assignment is desired, a classifier 314 may be coupled to monitor the queue usage (e.g. the space available in each queue). In the embodiment shown in Fig. 3, a classifier 314 serves to dynamically assign and/or reassign priorities for different session or data stream types. That is, the classifier may modify the priority classification in the look-up table 312 to reassign priorities. Other dynamic priority assignment schemes may be employed without deviating from one or more embodiments of the invention. In one embodiment, the classifier 314 is coupled to the frame processing scheduler 302 to permit the frame processing scheduler 302 to change the priority of frame types in the classifier 314. The classifier 314 also receives feedback from the priority queues 306, 308, and 310 so that it can modify session or data stream priorities as needed. The classifier 314 may serve to provide load balancing by modifying a session or stream priority to avoid overflow in one or more of the queues.

[0026] A switch 316 is couple to the priority queues to take new tasks from the priority queues 306, 308, and 310 and place them in a shared execution queue 318. In one implementation, the switch 316 places the new tasks in the shared execution queue 318 in the order in which they are read from the priority queues 306, 308, and 310.

[0027] This scheme does not change the interface to processing units (e.g. DSPs) that retrieve tasks from the shared execution queue 318. That is, whenever a processing unit is free it checks the shared execution queue 318 for new tasks to process. If a task is available, then it takes the task and starts processing it. However, one difference between the shared execution queue 318 of this embodiment of the invention and prior art shared execution queues is that, according to one aspect of this embodiment of the invention, tasks are arranged or sorted according to processing priority.

[0028] Another aspect of one embodiment of the invention reduces the probability of

overflow in the shared execution queue 318. The switch 316 may provide indirect feedback to the rest of the system as to the state of the shared execution queue 318. The switch 316 may delay from or stop removing new tasks from the priority queues 306, 308, and 310, if the shared execution queue 318 starts to get full or is filled beyond a threshold level. Delaying or stopping the transfer of new tasks from the priority queues to the shared execution queue 318 will cause some of the priority queues to start to get full or reach a threshold level. If this is the case, one embodiment of the classifier 314 may dynamically reassign session or data stream priorities to prevent overflow of priority queues. That is, if a particular priority queue starts to get full, session or data stream types may have their priority reassigned to route new tasks to other priority queues (e.g. priority queues with more available space, lower priority queues, etc.).

[0029] One aspect of one embodiment of the invention provides fair queuing of new tasks in the shared execution queue.

[0030] Figure 4 is a block diagram of various priority queues 402, 404, and 406, a switch 408, and a shared execution queue 410 illustrating fair apportionment of new tasks according to one embodiment of the invention. In this illustrative embodiment, three priority queues 402, 404, and 406, are employed to hold new tasks of different priority levels. Priority levels may be assigned according to the processing requirements of each task, session, and/or data stream type. In this example, the High Priority Queue 402 stores new tasks requiring five milliseconds processing times (e.g. the frame should be processed within five milliseconds, before the next frame arrives). Similarly, the Medium Priority Queue 404 holds new tasks requiring ten milliseconds processing times and the Low Priority Queue 406 holds new tasks requiring thirty milliseconds processing times.

[0031] To implement fair execution of new tasks held in the priority queues the switch 408 retrieves one task from each priority queue 402, 404, and 406. For example, for every Low Priority Queue task (L1) placed into the shared execution queue 410, one High Priority Queue task (H1) and one Medium Priority Queue task (M1) is placed into the shared execution queue 410.

[0032] Another aspect of one embodiment of the invention provides weighted fair

queuing in the shared execution queue of new tasks from all priority queues. That is, new tasks in a priority queue should be transferred in a fair and weighted manner relative to the tasks in other priority queues.

[0033] Figure 5 is a block diagram of various priority queues, a switch, and a shared execution queue illustrating fair weighted apportionment of new tasks according to one embodiment of the invention. In this illustrative embodiment, three priority queues 502, 504, and 506, are employed to hold new tasks of different priority levels. As in the example of Figure 4, priority levels are assigned according to the processing requirements of each task, session, or data stream type. Likewise, the High Priority Queue 502 stores new tasks requiring five milliseconds processing times, the Medium Priority Queue 504 holds new tasks requiring ten milliseconds processing times and the Low Priority Queue holds new task requiring thirty milliseconds processing times.

[0034] To implement fair and weighted execution of new tasks held in the priority queues the switch 508 retrieves tasks according to their relative processing requirements. For example, for every Low Priority Queue task (L1)(thirty millisecond task) placed into the shared execution queue 510, six High Priority Queue tasks (H1, H2, H3, H4, H5, and H6)(five millisecond tasks) and three Medium Priority Queue tasks (M1, M2, and M3) (ten millisecond tasks) are placed into the shared execution queue 510.

[0035] The number of new tasks retrieved from each priority queue in a given task retrieval cycle is directly related to the processing requirement of the tasks in each queue. For purposes of this description, a task retrieval cycle is defined as the period in which at least one new task is retrieved from every priority queue, assuming that new tasks are available in every priority queue. In the example illustrated in Fig. 5, new tasks are retrieved from each priority queue such that space in the shared execution queue is allotted equally according to processing time restrictions. For instance, in one task retrieval cycle, six new tasks (H1, H2, H3, H4, H5, and H6) from the High Priority Queue 502, each with a five millisecond processing time requirement (for a total of thirty milliseconds) are retrieved, three new tasks (M1, M2, and M3) from the Medium Priority Queue 504, each with a ten millisecond processing time requirement (for a total of thirty milliseconds) are retrieved, and one new tasks (L1) from the Low Priority Queue 506, with a thirty millisecond processing time

requirement is retrieved.

[0036] The order in which tasks are retrieved from the different priority queues in a given task retrieval cycle may vary without deviating from alternative embodiments of the invention. In the exemplary embodiment illustrated in Figure 5, the tasks from each priority queue are retrieved together in a given task retrieval cycle. That is, tasks H1, H2, H3, H4, H5, and H6 are retrieved, then tasks M1, M2, and M3 are retrieved, and lastly task L1 is retrieved.

[0037] In implementing a fair and weighted scheduling scheme, there are various ways in which the tasks from the various priority queues 502, 504, and 506 may be retrieved, organized, and/or ordered within the shared execution queue for a given task retrieval cycle. Generally, so long as tasks are retrieved from the various priority queues in fair and weighted manner, the order in which the tasks are arranged in the shared execution queue during a given task retrieval cycle does not deviate or vary from one embodiment of the invention.

[0038] Figure 6 illustrates various other ways in which new tasks may be queued in a shared execution queue according to a weighted scheduling scheme. In one shared execution queue 510', new tasks from the various priority queues are substantially evenly distributed (e.g. H1, H2, M1, H3, H4, M2, H5, H6, M3, L1) for a given task retrieval cycle. In another shared execution queue 510", new tasks from the various priority queues are retrieved in reverse order of priority (L1, M1, M2, M3, H1, H2, H3, H4, H5, H6) for a given task retrieval cycle. In yet another shared execution queue 510"', new tasks are retrieved in groups of no more than three new tasks from any given priority queue at one time for a given task retrieval cycle.

[0039] In yet another implementation of a weighted scheduling scheme, the tasks in a shared execution queue 510"" are retrieved in a random or pseudo-random order (e.g. M1, M2, H1, L1, H2, H3, H4, M3, H5, H6) from the various priority queues for a given task retrieval cycle. While the new tasks may randomly arranged/retrieved during a given retrieval cycle, the weighted scheduling relationship should be maintained during any given retrieval cycle (e.g. six high-priority tasks, three medium-priority tasks, and one low-priority task).

[0040] Note that fair and weighted task scheduling of new tasks applies when there are sufficient tasks available in the various queues employed. When in a given task retrieval cycle one or more queues do not have a sufficient number of new tasks to be placed into the shared execution queue in a weighted manner, then various different schemes may be employed during that retrieval cycle. In one implementation, the switch merely retrieves a new task from another priority queue instead, and then continues with the fair and weighted task scheduling as before.

[0041] Figures 7 and 8 illustrate a couple of ways in which tasks from one priority queue may be used to fill-in for a shortage of tasks from another priority queue. In one embodiment, illustrated in Figure 7, the switch 708 may retrieve a new task from the next lowest priority queue. For instance, as shown in the shared execution queue 710, when the High Priority Queue 702 does not have any more new tasks, task M1* from the Medium Priority Queue 704 is retrieved by the switch 708 instead. The switch 708 then continues with the fair and weighted task scheduling as before.

[0042] In another embodiment, illustrated in Figure 8, the switch 808 may retrieve the new task from the next highest priority queue. For examples, as shown in the shared execution queue 810, when the Medium Priority Queue 804 does not have any more new task, task H7* from the High Priority Queue 802 is retrieved by the switch 808 instead. The switch 808 then continues with the fair and weighted task scheduling as before.

[0043] In another implementation, illustrated in Figure 9, when a particular priority queue does not have new tasks to be processed, the switch 908 does not retrieve a new task from any other priority queue as a replacement and, instead, just continues its fair and weighted task scheduling by moving to the next priority queue. For example, as shown in the shared execution queue 910, when there are no more high priority tasks (High Priority Queue 902 only has five new tasks instead of six), the switch merely moves on to the next priority queue (Medium Priority Queue 904 in this example) in the weighted scheduling sequence and retrieves M1 after H5. The switch 908 then continues with the fair and weighted task scheduling as before.

[0044] While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely

illustrative of and not restrictive on the broad aspects of various embodiments of the invention, and that these embodiments not be limited to the specific constructions and arrangements shown and described, since various other modifications are possible. Additionally, it is possible to implement the embodiments of the invention or some of their features in hardware, programmable devices, firmware, software or a combination thereof.